Lesson 16: OOP #6, Wrap-Up (W05D1)

Balboa High School

Michael Ferraro

September 14, 2015

Do Now

- Identify a partner in the class with whom you'll work on exercises to prepare you for next class' quiz.
- You and your partner should be at the same workstation.
- Proceed on to the following slides and complete the exercises.
- Once finished, work on Problem Set #2, which is due on Wed., 9/16, by the start of 5^{th} Period.

Aim

Students will practice working with OOP and continue working on PS #2.

New Project

- In Eclipse, create a new project called `VehicleOOP`.

Class Vehicle

- Create class `Vehicle` with the following `private` fields.

```
numWheels : int
mpg       : int        //miles per gallon
maker     : String     //brand, e.g., Ford
```

Driver Class

- Create a new class in the project called `VehicleDriver`.

Driver Class

- Create a new class in the project called `VehicleDriver`.
- A `VehicleDriver` is not a kind of `Vehicle`, so do not use `extends`!

Make a `Vehicle`

- In the driver class, have the following actions occur, in the given order.
  1. Create an instance of `Vehicle` called `v1`.
  2. Set `numWheels` to 3.
  3. Set `mpg` to 39.
  4. Set `maker` to `Yamaha`.

Make a `Vehicle`

- In the driver class, have the following actions occur, in the given order.
  1. Create an instance of `Vehicle` called `v1`.
  2. Set `numWheels` to 3.
  3. Set `mpg` to 39.
  4. Set `maker` to `Yamaha`.
- **NOT noticing a problem?** Go back to the slide that described the field variables in `Vehicle` and see which detail you missed.

Make a `Vehicle`

- In the driver class, have the following actions occur, in the given order.
  1. Create an instance of `Vehicle` called `v1`.
  2. Set `numWheels` to 3.
  3. Set `mpg` to 39.
  4. Set `maker` to `Yamaha`.
- **NOT noticing a problem?** Go back to the slide that described the field variables in `Vehicle` and see which detail you missed.
- **You ARE noticing a problem?** Write appropriate *mutator methods* in `Vehicle`.

Add Accessor Methods

- Add the following accessor methods to the `Vehicle` class.
  - `getNumWheels()`
  - `getMpg()`
  - `getMaker()`

Create a `toString()` Method

Add the following method to the `Vehicle` class.

```
public String toString() {
    String desc;
    desc  = "This vehicle has " + numWheels + " wheels,\n";
    desc += "gets " + mpg + " miles to the gallon, and\n";
    desc += "is manufactured by " + maker + ".";

    return desc;
}
```

Using `toString()`

- Add this statement to the driver class after you've set `v1`'s values —

  `System.out.println( v1.toString() );`

  — and make sure you see `v1`'s description printed.

Using `toString()`

- Add this statement to the driver class after you've set `v1`'s values —

  ```
  System.out.println( v1.toString() );
  ```

  — and make sure you see `v1`'s description printed.

- Now try printing the object itself. Replace the line you added from above with this one:

  ```
  System.out.println( v1 );
  ```

Using `toString()`

- Add this statement to the driver class after you've set `v1`'s values —

  ```
  System.out.println( v1.toString() );
  ```

  — and make sure you see `v1`'s description printed.

- Now try printing the object itself. Replace the line you added from above with this one:

  ```
  System.out.println( v1 );
  ```

- Whenever an object's class provides a `toString()` method, that method will be called whenever someone tries to "print" the object itself!

Making a Child (Class)

- Add a new class to the project called `Car`. Make it a subclass of `Vehicle`.

Making a Child (Class)

- Add a new class to the project called `Car`. Make it a subclass of `Vehicle`.
- This new class should have the following fields *in addition* to the ones inherited from the superclass:
  - `numDoors : int`
  - `model : String //e.g., Mustang or Forester`

Making a Child (Class)

- Add a new class to the project called `Car`. Make it a subclass of `Vehicle`.
- This new class should have the following fields *in addition* to the ones inherited from the superclass:
  - `numDoors : int`
  - `model : String //e.g., Mustang or Forester`
- Create appropriate accessor and mutator methods for `Car`'s fields.

Instantiate `Car` into an Object

- In `VehicleDriver`, create an instance of `Car` called `car1`.
- Using the available mutator methods, set *all* available *instance variables*[1] — except `numWheels` — to the values of your choice. `numWheels` should not be set by you.
- Print out the description of `car1`.

[1] I'm purposely using *instance variables* and *fields* interchangably so you get used to these synonyms. The same goes for terminology for creating objects: *instantiating. . . into an object, creating an instance of. . .* , etc.

A Better Constructor

- Currently, your `Vehicle` and `Car` classes have default, no-args constructors provided by Java.
- Add a functional constructor to `Car.java`:[2]

```
public Car() {
    //assume all Cars have 4 wheels
    numWheels = 4;
}
```

- Identify and fix the problem with this constructor. Make sure you understand exactly why there is a problem!

[2] Place the constructor below the fields and above the methods.

Overriding a Superclass' Method

- When a `Car`'s `toString()` method is called, the output is in the same format as all other `Vehicles`.
- Add a `toString()` method to `Car` that is more interesting than the original. Prove that this new `toString()` is called when you print your `Car`'s description.

Rest of the Period...

- For the rest of the period, work on PS #2.
- Leverage the course website as a means of getting help and giving hints to your peers in need!

HW

Continue working on PS #2.